I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being transmitted via the Office electronic filing system in accordance with § 1.6(a)(4).

Dated: November 30, 2009
Electronic Signature for Kevin J. Canning: / Kevin J. Canning/

Docket No.: MWS-070RCE3
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Vijay Raghavan *et al.*

Application No.: 09/855,199 | Confirmation No.: 8175

Filed: May 14, 2001 | Art Unit: 2128

For: GRAPHICAL FUNCTIONS | Examiner: S. A. Alhija

ARGUMENTS FOR PRE-APPEAL BRIEF REQUEST FOR REVIEW

MS AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

The following is submitted together with a Notice of Appeal under 37 C.F.R. §41.31 and in support of a Pre-Appeal Brief Request for Review in the above-identified Application.

At issue in this appeal is a new capability made available in graphical modeling environments, especially in statecharts. Embodiments of claimed invention allow a user to graphically define a function having a function prototype. The function may then be called textually from within the statechart according to the function prototype. Doing so is neither disclosed nor rendered obvious by the combination of Kodosky (Publication No. 2002/0083413) and Toeppe (*Practical Validation of Model Based Code Generation for Automotive Applications*), the two references cited by the Examiner.

The Examiner has not made a prima facie case of obviousness of independent claims. In particular, the claims recite "the function that is represented graphically has a function prototype that specifies a syntax for invoking the function " and "calling the function that is represented graphically by the function name according to the syntax specified by the function prototype from within the graphical representation of the finite state machine." The Examiner impermissibly equates the cited references' textual functions or junctions in a statechart diagram with the graphical functions being claimed by Applicants, even though the claims explicitly

recite functions that are represented graphically. Further, the cited references do not call the function that is represented graphically by the function name, as recited in the claims.

Representative claim 1 recites a computer-implemented method comprising:

> providing a graphical user interface for defining a function to be used in a graphical representation of a finite state machine, where the graphical representation is an executable model of the finite state machine in a statechart system and includes at least one state and at least one transition;
>
> *representing the function graphically such that the function is graphically represented separately from the at least one state and at least one transition in the graphical representation of the finite state machine, wherein* ***the function that is represented graphically has a function prototype that specifies a syntax for invoking the function, the function prototype specifying a function name for the function***, and the function is defined in the statechart system in a graphical language; and
>
> ***calling the function that is represented graphically by the function name according to the syntax specified by the function prototype from within the graphical representation of the finite state machine.***

Kodosky and Toeppe references are silent with respect to at least the above boldfaced features of claim 1, and the variations thereof found in the other independent claims.

The present Application is directed to graphical representations of functions that may be described, defined, represented, or invoked in a modeling system for finite state machines. A finite state machine is a representation of an event-driven system. In an event-driven system, the system transitions from one state to another prescribed state based on events and conditions. A state has a label that specifies its name and actions. Actions take place during execution of a statechart system. An action may be executed either as part of a transition from one state to another or based on the activity status of a state. In a traditional statechart system, a built-in action language defines the types of actions that may be specified and their associated notations. An action can be a function call to a built-in function or to a textual function defined by a user.

An embodiment of a statechart system called StateFlow™ from the Mathworks, Inc. – the assignee of the present application – also has connective junctions, which are decision points in the system. A connective junction is represented by a circle and is a graphical object that simplifies diagram representations and is used to represent the flow of code structure.

The embodiments of the claimed invention allow a user to represent graphically a procedure performed by a function in a statechart system. Such a graphically-represented function provides a number of advantages, such as the ability to leverage the native parser of the statechart environment, rather than an external parser. Unlike textual functions, such as C and

MATLAB® functions, graphical functions are defined using a graphical editor and they may reside in statechart systems along with the diagrams that invoke them, which makes graphical functions easier to create, access and manage than textual functions, whose creation requires external tools and whose definitions reside separately from the model.

Kodosky is one example of an environment that utilizes externally-defined textual code (see, e.g., Kodosky at paragraphs [0168]-[0169]). Kodosky is generally directed to a system and method for generating a graphical program in response to state diagram information (Kodosky at Abstract). Kodosky includes a state diagram that specifies a plurality of states and state transitions. A graphical program generator generates a framework for source code that includes placeholders a user fills in with code for states and transitions (Kodosky at Abstract).

Toeppe is generally directed to the validation of C code generated from a Computer Aided Control System Design (CACSD) environment. (Toeppe at §1, second paragraph, and §4.4). Toeppe presents some screen shots from an older version of Stateflow™ modeling environment that doesn't support graphical functions. These screen shots illustrate the components of conventional state diagrams: states, transitions and conditions that must be satisfied to cause the transitions. No graphical functions are defined in these conventional state diagrams.

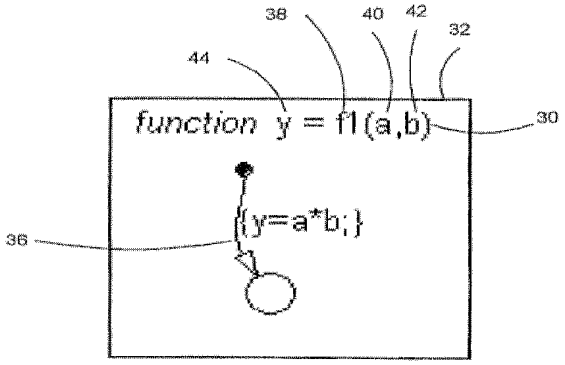
1. Neither Kodosky nor Toeppe disclose or suggest that *the function that is represented graphically has a function prototype that specifies a syntax for invoking the function, the function prototype specifying a function name for the function.*

The Examiner correctly recognizes that Kodosky does not disclose that *the function that is represented graphically has a function prototype that specifies a syntax for invoking the function, the function prototype specifying a function name for the function* (Office Action at page 4). Instead, the Examiner relies on Toeppe for the above-quoted feature of claims 1, 5, 12, 17, 24, 32, 34, 43, 52, 61, 66, 70, and 74.

While Toeppe mentions the Stateflow™ statechart modeling environment, Toeppe's treatment of it is cursory and it merely suggests that verifying code generated from the Stateflow™ diagrams may prove difficult. (Toeppe, §6.1.2) The Examiner cites §§ 4.4 and 6.1, as well as Figures 6-10 of Toeppe as disclosing a graphically defined function with a function prototype. However, none of the cited figures show a graphical function with a function prototype. The figures focus on junctions and the complications they may provide in verifying code generated from Stateflow™ diagrams. Stateflow™ modeling environment is provided by

the assignee of the present application and is very familiar to the Applicants. Graphical functions were not present in the Stateflow™ modeling environment at the time of publication of Toeppe. Graphical functions were first made available to the public as part of Stateflow™ on or after May 20, 2000, less than a year prior to filing of this application, as further explained in the Declaration Pursuant to 37CFR 1.132 of Vijaya Raghavan, submitted on 12/5/2008.

The most that can be noted in the cited Figures of Toeppe is that states have names and transitions have conditions associated with them. The following figures from the present application and from Toeppe illustrate the differences:



A graphical function as depicted in Figure 2 of the present Application

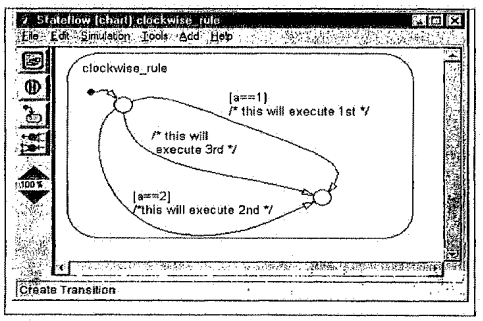


Figure 5 Example of Clockwise Rule and Precedence for Transitions

Figure 5 of Toeppe

As shown above, the graphical function 32 of the present application has a function name 44 and a function prototype 30 that specifies the name of the function 38 ("f1"), the type and number of its inputs 40, 42 ("a" and "b") and the type of its output 44 ("y"). This graphical function may be invoked textually from within the statechart multiple times with different arguments, by calling this function according to the syntax specified in its function prototype.

In comparison, shown in Toeppe is a statechart named "clockwise_rule" with two junctions and transitions between those junctions. There is no function prototype and "clockwise_rule" may not be called from within another statechart by its name.

Sections 4.4 and 6.1 of Toeppe do mention function prototypes; however, in both instances, Toeppe is referring to functions in C source code, and not a function prototype for *the function that is represented graphically*. At §4.4, Toeppe explicitly says that the function parameters are used for controlling the modularity or partitioning "of the C source" (Toeppe at §4.4, first four lines). At §6.1, Toeppe mentions a "complete function prototype" in the section

entitled "Auxiliary Functionality," which is distinct from the section labeled "Stateflow Blocks" (Toeppe at §6.1). Indeed, Toeppe does not represent any function graphically, and so Toeppe cannot disclose a function prototype for a function that is represented graphically, nor does it suggest such a prototype.

2. Neither Kodosky nor Toeppe discloses or suggests *calling the function that is represented graphically by the function name according to the syntax specified by the function prototype from within the graphical representation of the finite state machine*

The Office Actions appear to include a contradiction regarding this feature of independent claims: although the Examiner correctly recognizes that Kodosky does not disclose a function prototype that specifies a syntax or a function name, the Examiner also asserts that Kodosky does call a graphically represented function "according to the syntax specified by the function prototype" (Office Action made Final at page 4, 3rd paragraph). Kodosky cannot fail to disclose a function prototype and yet call such a function by the prototype at the same time.

Toeppe also does not disclose or suggest this feature of claim 1. As noted above, the "functions" referred to in Toeppe are textual functions in the C source. Toeppe's functions are neither *a function that is represented graphically*, nor are they provided *in the graphical representation of the finite state machine*, nor are they called *from within the graphical representation of the finite state machine*, as recited in claim 1. Accordingly, Toeppe does not disclose or suggest *calling the function that is represented graphically by the function name according to the syntax specified by the function prototype from within the graphical representation of the finite state machine.*

Therefore, neither of the references include graphical functions with function prototypes, syntaxes, function names, and neither of the cited references call the cited elements by function name. Those are fundamental differences between the claimed invention and the cited references. Accordingly, Applicants respectfully request withdrawal of the rejections and the allowance of all claims.

Dated: November 30, 2009

Respectfully submitted,

Electronic signature: /Kevin J. Canning/
Kevin J. Canning
Registration No.: 35,470
LAHIVE & COCKFIELD, LLP
One Post Office Square
Boston, Massachusetts 02109-2127
Attorney/Agent For Applicant